# Matematička teorija računarstva

## Predavanja 14

Matko Botinčan

PMF – Matematički odjel

31.01.2007.

# Izvori materijala za ovo predavanje

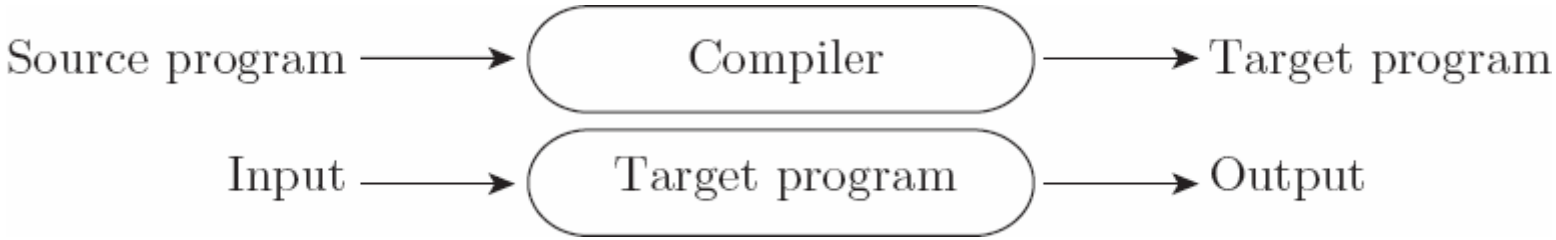
- X. Jiao – CS 3240: Languages and Computation
  - http://www-static.cc.gatech.edu/classes/AY2007/cs3240_fall/
- T. K. Prasad – CS 780: Compiler Design and Construction I
  - http://www.cs.wright.edu/~tkprasad/courses/cs780/cs780.html
- J. Riely – CSC347/447: Concepts of Programming Languages
  - http://condor.depaul.edu/~jriely/csc447winter2007/index.html
- D. Stotts – COMP 144: Programming Language Concepts
  - http://www.cs.unc.edu/~stotts/COMP144/
- Q. Yi – CS5363: Programming Languages and Compilers
  - http://www.cs.utsa.edu/~qingyi/cs5363/

# Prevođenje programa

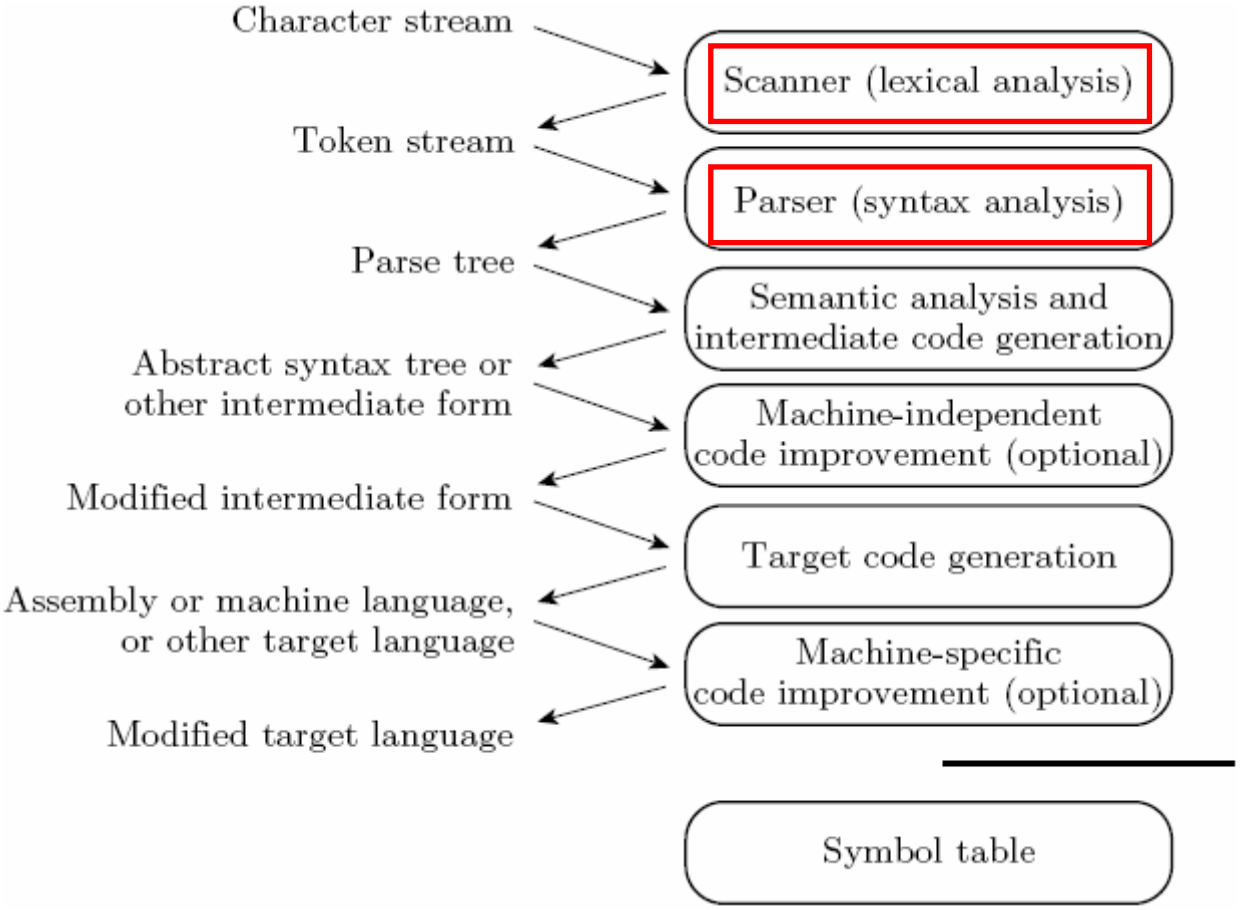
```
program gcd(input, output);
var i, j: integer;
begin
        read(i, j);
        while i <> j do
                if i > j then i := i – j;
                else j := j – i;
        writeln(i)
end.
```

```
27bdffd0 afbf0014 0c1002a8 00000000 0c1002a8 afa2001c 8fa4001c
00401825 10820008 0064082a 10200003 00000000 10000002 00832023
00641823 1483fffa 0064082a 0c1002b2 00000000 8fbf0014 27bd0020
03e00008 00001025
```

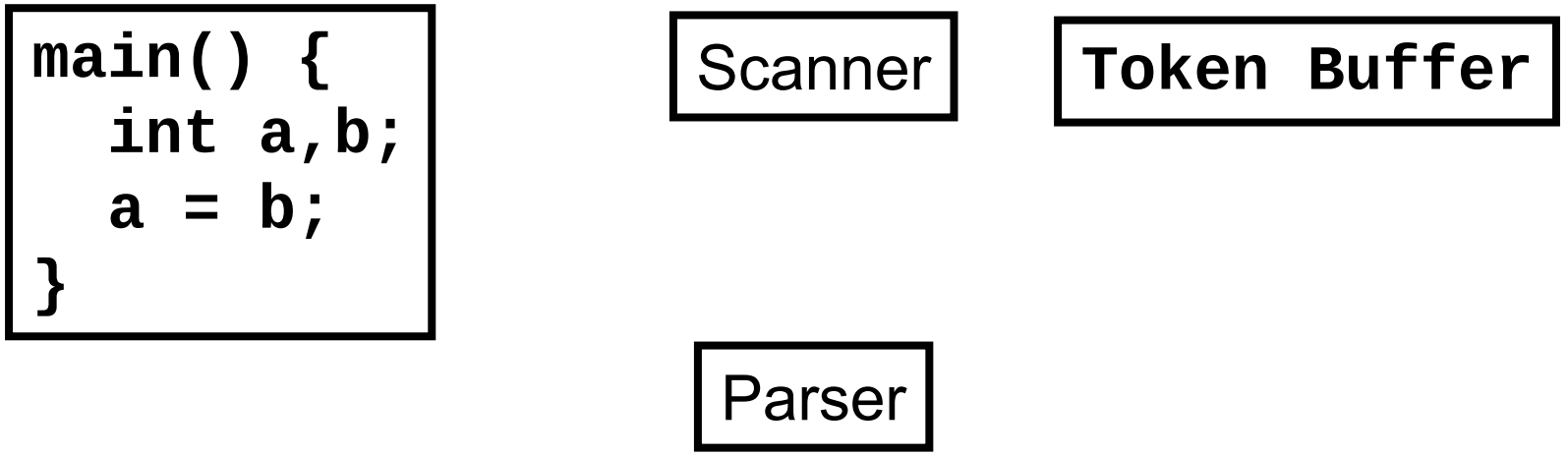
# Prevođenje programa



# Faze prevođenja programa

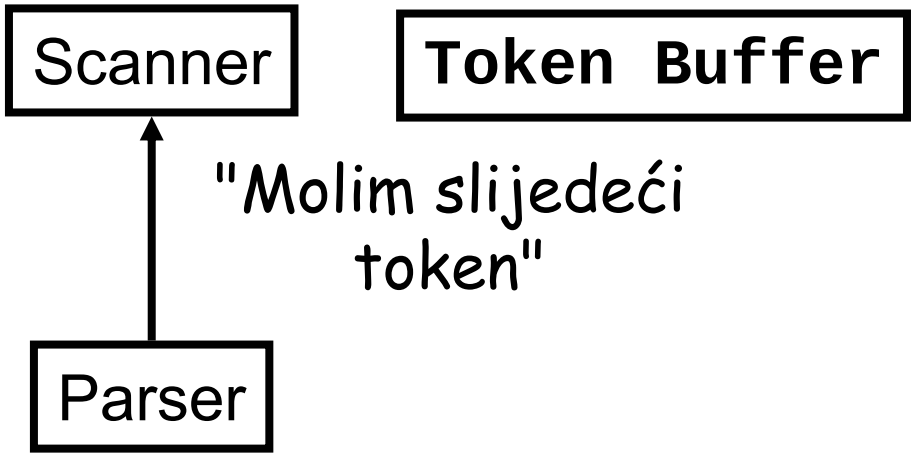


# Scanner/Parser

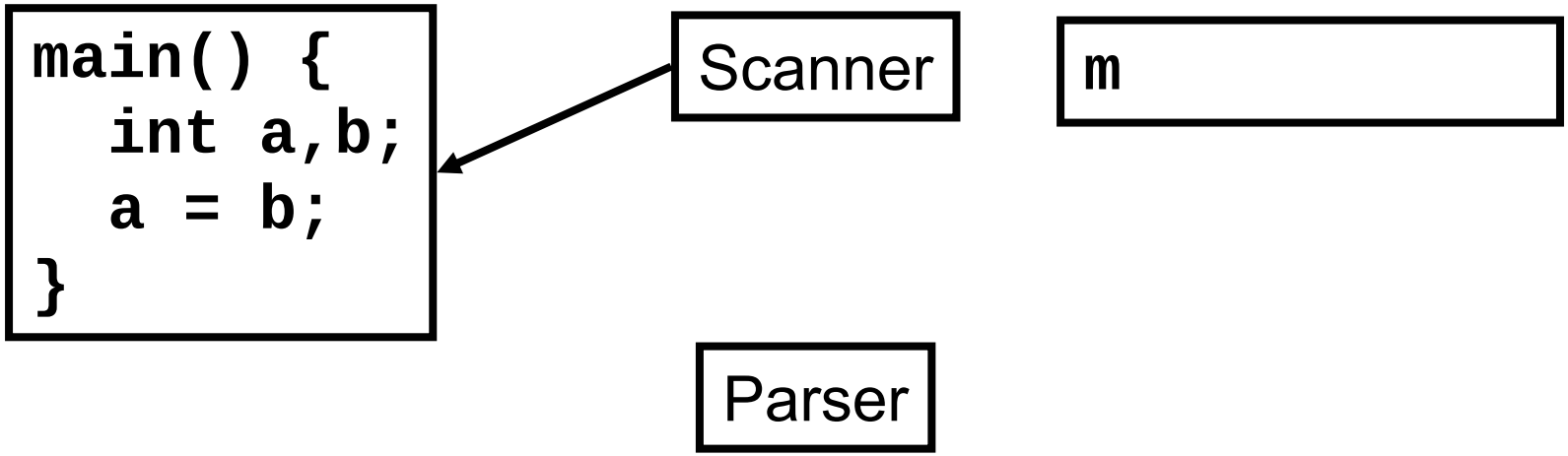


# Scanner/Parser

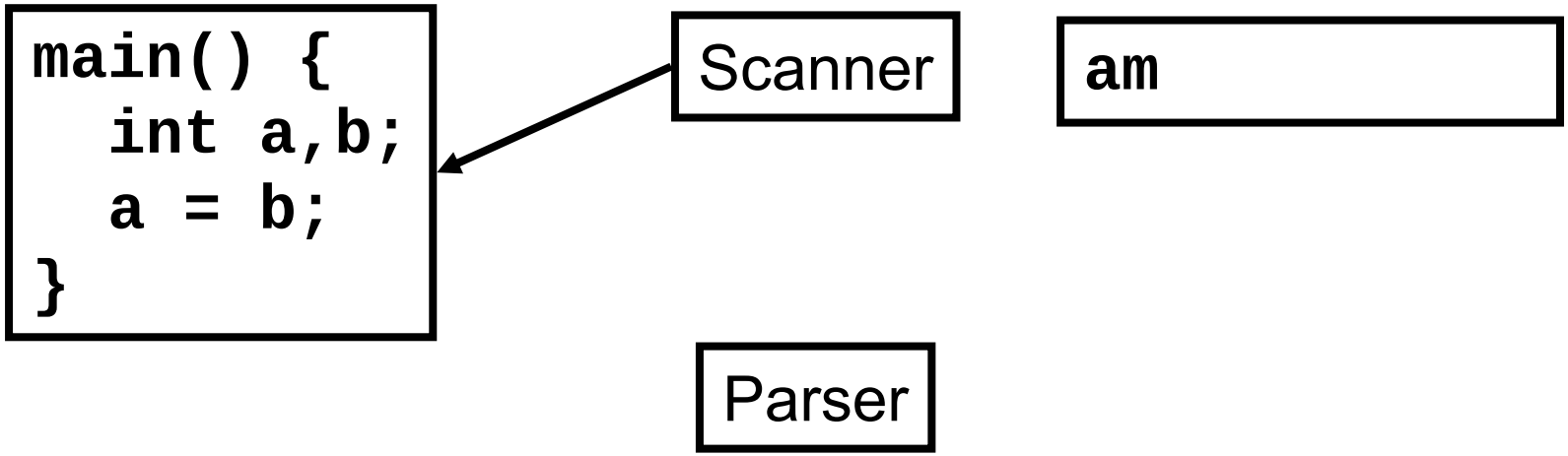
```
main() {
  int a,b;
  a = b;
}
```



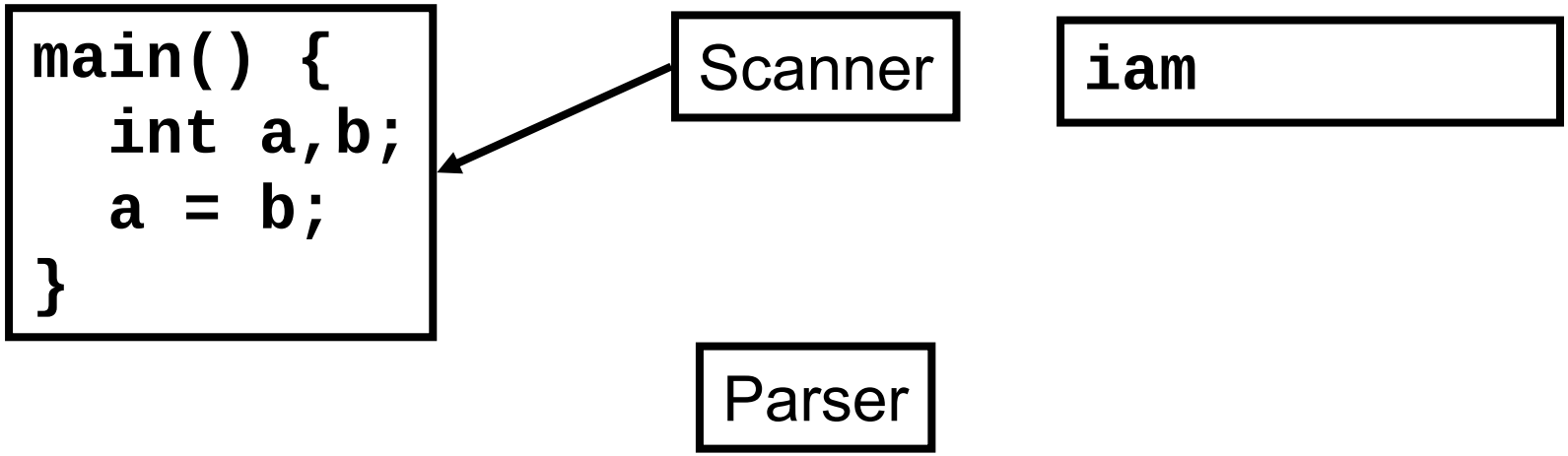
# Scanner/Parser



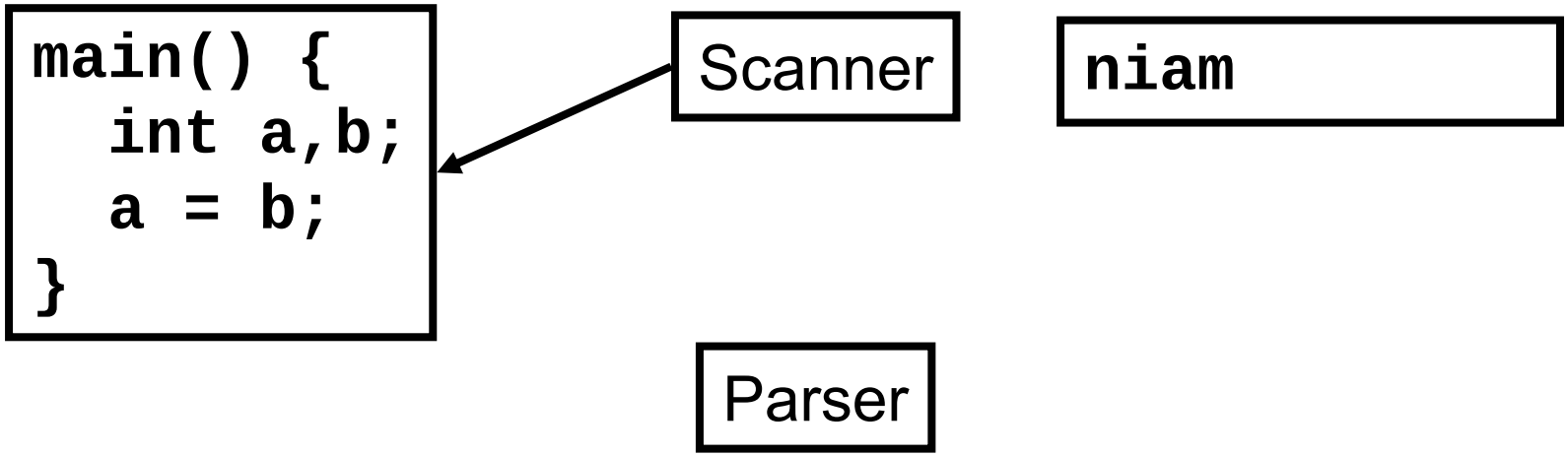
# Scanner/Parser



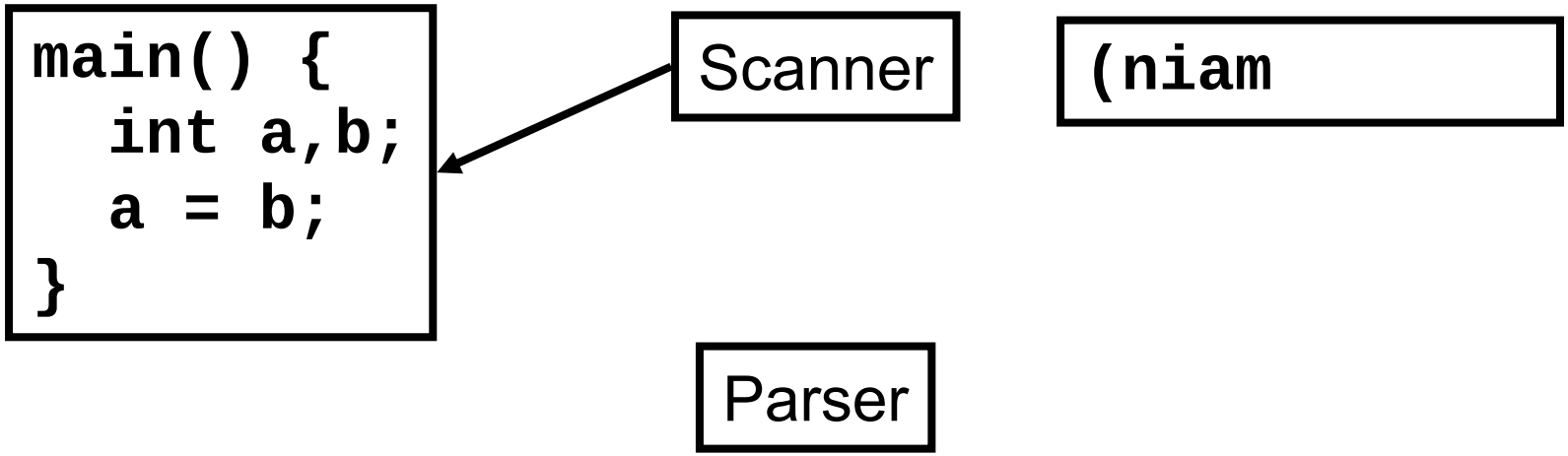
# Scanner/Parser



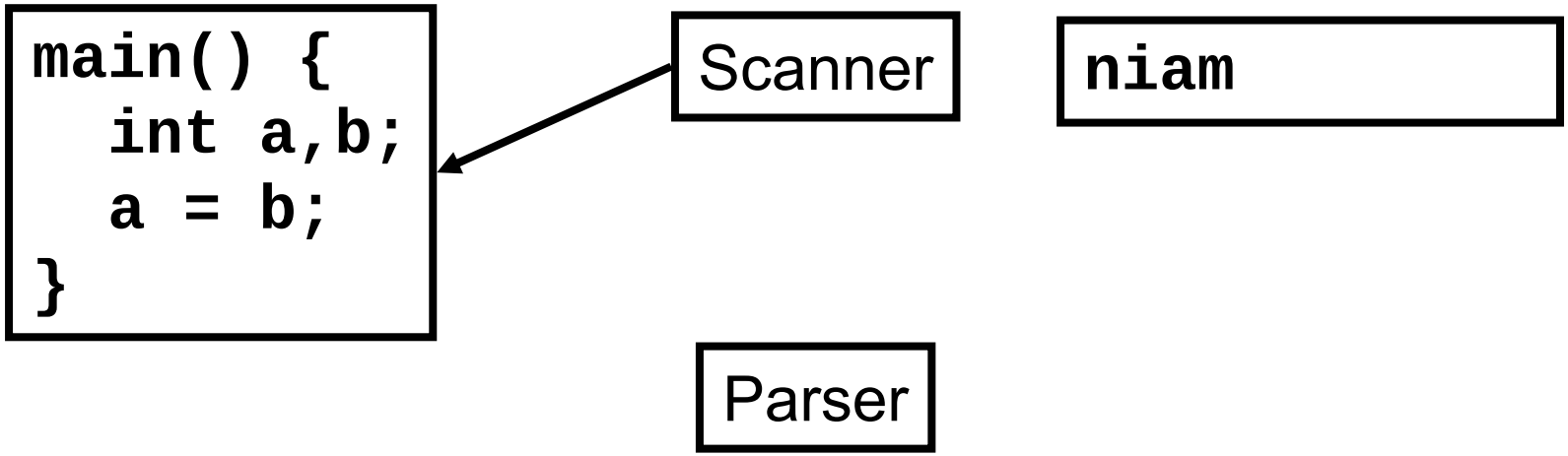
# Scanner/Parser



# Scanner/Parser

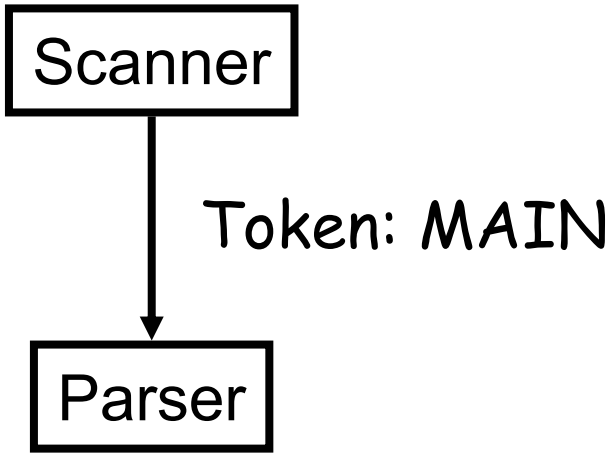


# Scanner/Parser



# Scanner/Parser

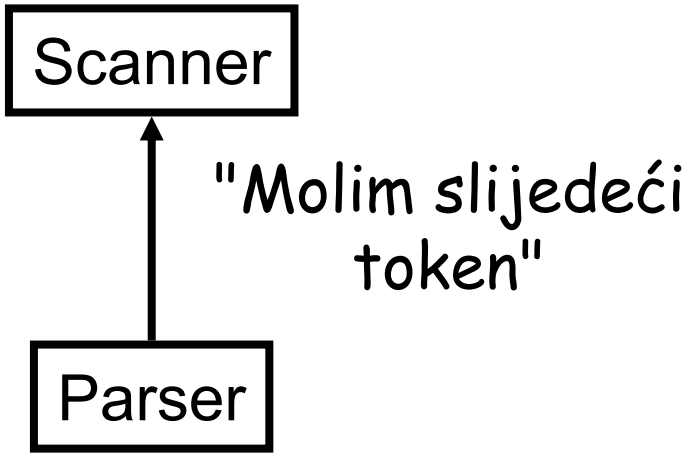
```
main() {
   int a,b;
   a = b;
}
```




**<C-PROG> → MAIN OPENPAR <PARAMETERS> CLOSEPAR <MAIN-BODY>**

# Scanner/Parser

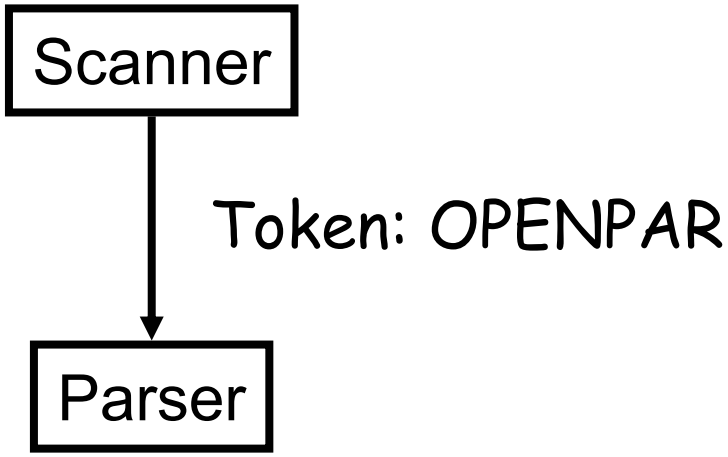
```
main() {
   int a,b;
   a = b;
}
```




**<C-PROG> → MAIN OPENPAR <PARAMETERS> CLOSEPAR <MAIN-BODY>**

# Scanner/Parser

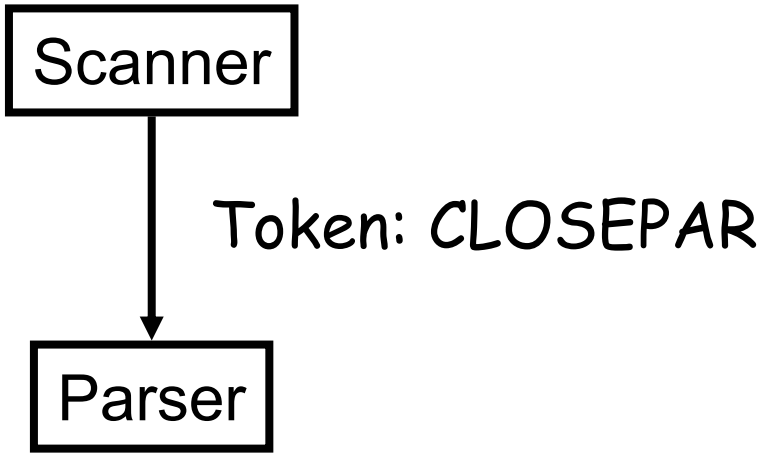
```
main() {
   int a,b;
   a = b;
}
```




**<C-PROG> → MAIN OPENPAR <PARAMETERS> CLOSEPAR <MAIN-BODY>**

# Scanner/Parser

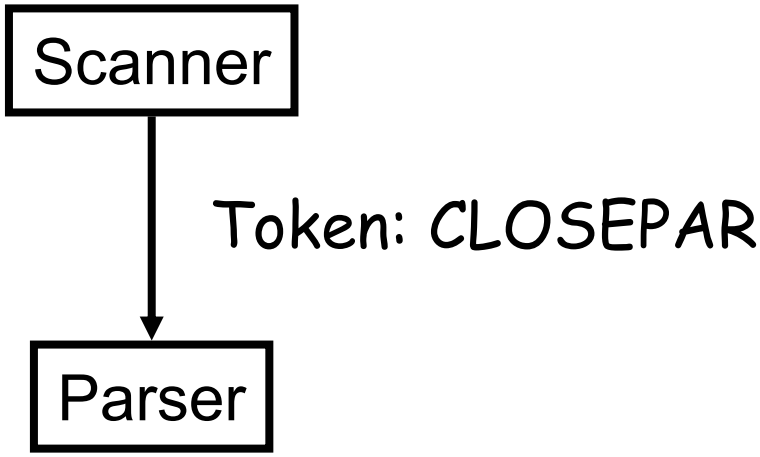
```
main() {
  int a,b;
  a = b;
}
```




<C-PROG> → MAIN OPENPAR <PARAMETERS> CLOSEPAR <MAIN-BODY>
<PARAMETERS> → NULL

# Scanner/Parser

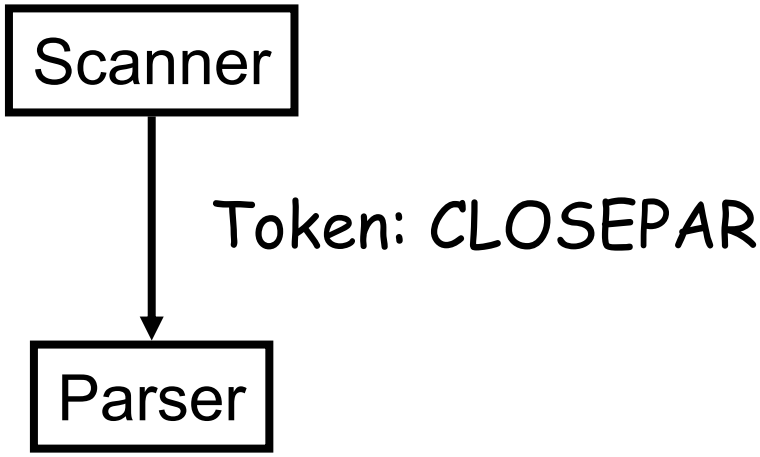
```
main() {
  int a,b;
  a = b;
}
```




**<C-PROG>** → **MAIN OPENPAR <PARAMETERS> CLOSEPAR <MAIN-BODY>**
**<PARAMETERS>** → **NULL**

# Scanner/Parser

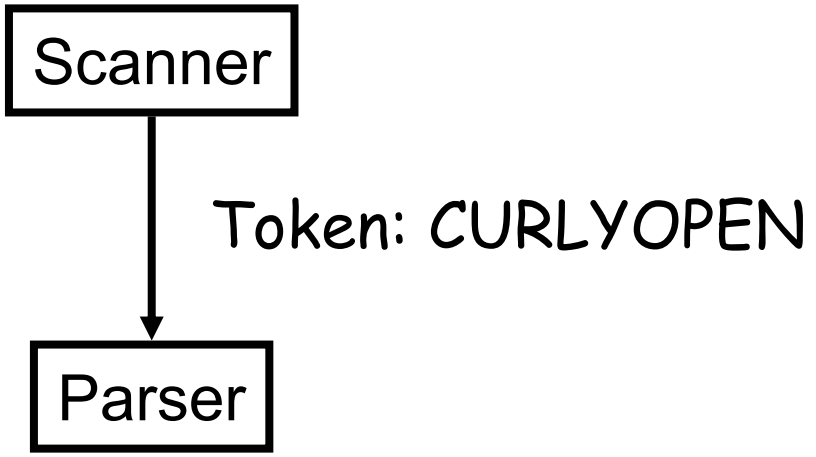
```
main() {
  int a,b;
  a = b;
}
```




**<C-PROG>** → **MAIN OPENPAR <PARAMETERS> CLOSEPAR <MAIN-BODY>**

# Scanner/Parser

```
main() {
  int a,b;
  a = b;
}
```

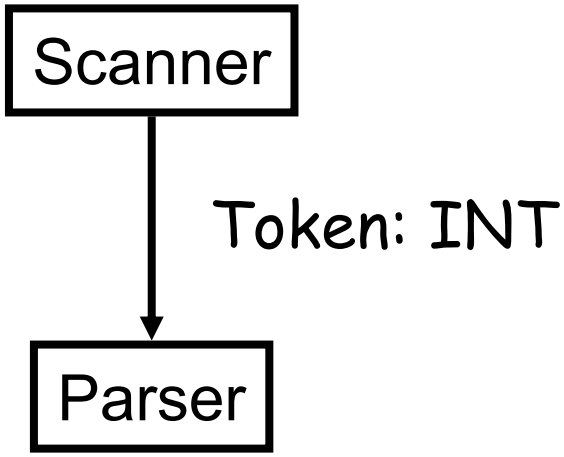



**<C-PROG> → MAIN OPENPAR <PARAMETERS> CLOSEPAR <MAIN-BODY>**

**<MAIN-BODY> → CURLYOPEN <DECL-STMT> <ASSIGN-STMT> CURLYCLOSE**

# Scanner/Parser

```
main() {
   int a,b;
   a = b;
}
```




<C-PROG> → MAIN OPENPAR <PARAMETERS> CLOSEPAR <MAIN-BODY>

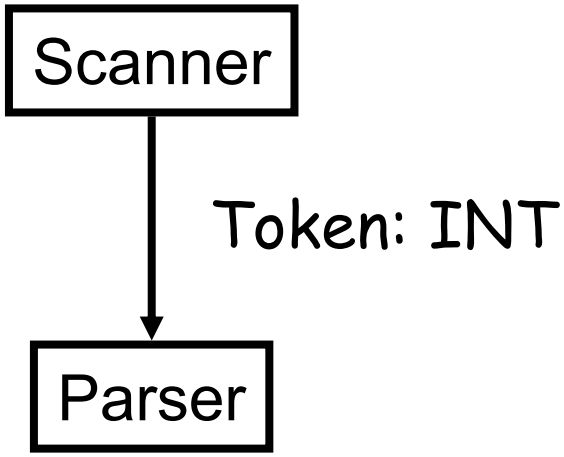
<MAIN-BODY> → CURLYOPEN <DECL-STMT> <ASSIGN-STMT> CURLYCLOSE

<DECL-STMT> → <TYPE>VAR<VAR-LIST>;

<TYPE> → INT

# Scanner/Parser

```
main() {
   int a,b;
   a = b;
}
```




<C-PROG> → MAIN OPENPAR <PARAMETERS> CLOSEPAR <MAIN-BODY>

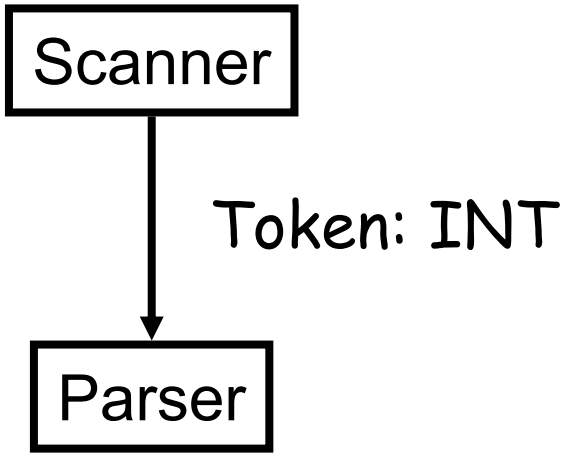
<MAIN-BODY> → CURLYOPEN <DECL-STMT> <ASSIGN-STMT> CURLYCLOSE

<DECL-STMT> → <TYPE>VAR<VAR-LIST>;

<TYPE> → INT

# Scanner/Parser

```
main() {
   int a,b;
   a = b;
}
```




<C-PROG> → MAIN OPENPAR <PARAMETERS> CLOSEPAR <MAIN-BODY>

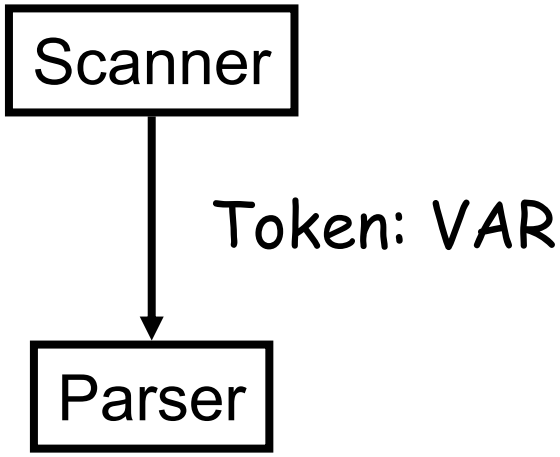
<MAIN-BODY> → CURLYOPEN <DECL-STMT> <ASSIGN-STMT> CURLYCLOSE

<DECL-STMT> → <TYPE>VAR<VAR-LIST>;

<TYPE> → INT

# Scanner/Parser

```
main() {
  int a,b;
  a = b;
}
```




<C-PROG> → MAIN OPENPAR <PARAMETERS> CLOSEPAR <MAIN-BODY>

<MAIN-BODY> → CURLYOPEN <DECL-STMT> <ASSIGN-STMT> CURLYCLOSE

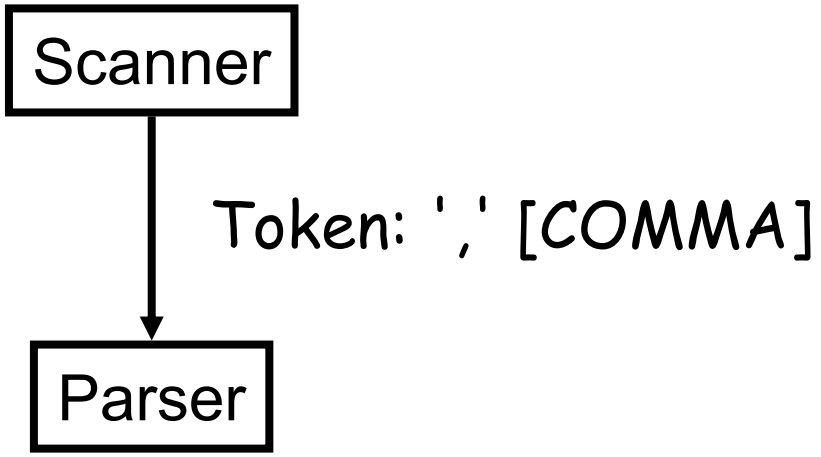
<DECL-STMT> → <TYPE>VAR<VAR-LIST>;

<VARLIST> → , VAR <VARLIST>

<VARLIST> → NULL

# Scanner/Parser

```
main() {
  int a,b;
  a = b;
}
```




<C-PROG> → MAIN OPENPAR <PARAMETERS> CLOSEPAR <MAIN-BODY>

<MAIN-BODY> → CURLYOPEN <DECL-STMT> <ASSIGN-STMT> CURLYCLOSE

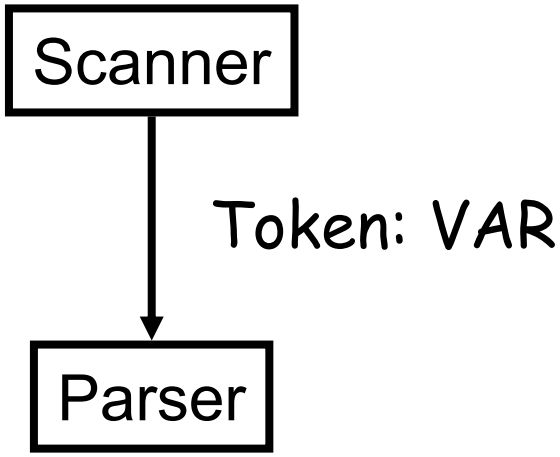
<DECL-STMT> → <TYPE>VAR<VAR-LIST>;

<VARLIST> → , VAR <VARLIST>

<VARLIST> → NULL

# Scanner/Parser

```
main() {
  int a,b;
  a = b;
}
```




<C-PROG> → MAIN OPENPAR <PARAMETERS> CLOSEPAR <MAIN-BODY>

<MAIN-BODY> → CURLYOPEN <DECL-STMT> <ASSIGN-STMT> CURLYCLOSE

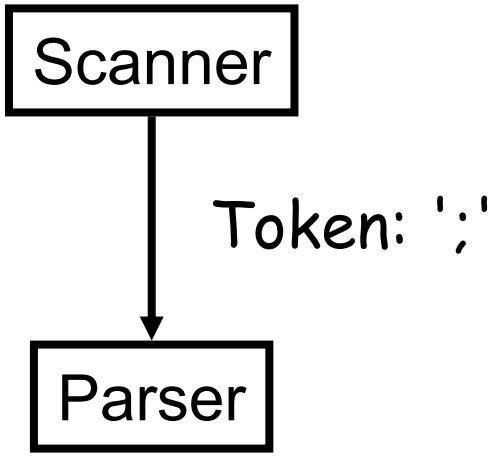
<DECL-STMT> → <TYPE>VAR<VAR-LIST>;

<VARLIST> → , VAR <VARLIST>

<VARLIST> → NULL

# Scanner/Parser

```
main() {
  int a,b;
  a = b;
}
```




<C-PROG> → MAIN OPENPAR <PARAMETERS> CLOSEPAR <MAIN-BODY>

<MAIN-BODY> → CURLYOPEN <DECL-STMT> <ASSIGN-STMT> CURLYCLOSE

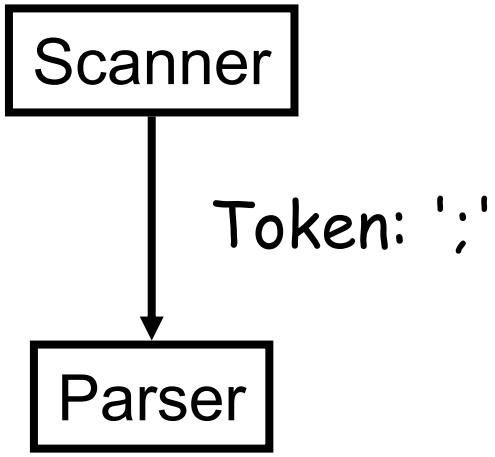
<DECL-STMT> → <TYPE>VAR<VAR-LIST>;

<VARLIST> → , VAR <VARLIST>

<VARLIST> → NULL

# Scanner/Parser

```
main() {
  int a,b;
  a = b;
}
```




<C-PROG> → MAIN OPENPAR <PARAMETERS> CLOSEPAR <MAIN-BODY>

<MAIN-BODY> → CURLYOPEN <DECL-STMT> <ASSIGN-STMT> CURLYCLOSE

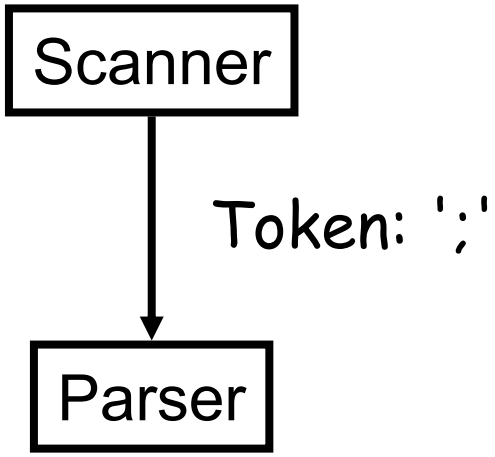
<DECL-STMT> → <TYPE>VAR<VAR-LIST>;

<VARLIST> → , VAR <VARLIST>

<VARLIST> → NULL

# Scanner/Parser

```
main() {
  int a,b;
  a = b;
}
```




<C-PROG> → MAIN OPENPAR <PARAMETERS> CLOSEPAR <MAIN-BODY>

<MAIN-BODY> → CURLYOPEN <DECL-STMT> <ASSIGN-STMT> CURLYCLOSE

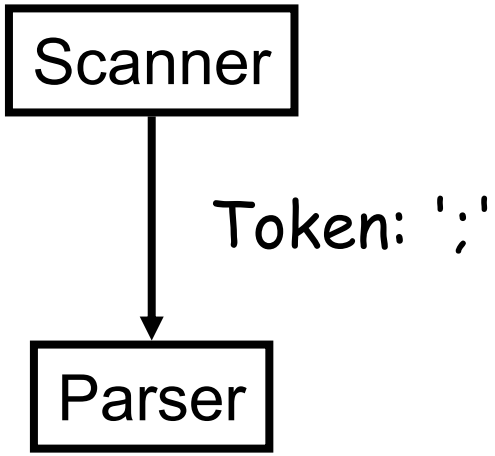
<DECL-STMT> → <TYPE>VAR<VAR-LIST>;

<VARLIST> → , VAR <VARLIST>

<VARLIST> → NULL

# Scanner/Parser

```
main() {
   int a,b;
   a = b;
}
```




<C-PROG> → MAIN OPENPAR <PARAMETERS> CLOSEPAR <MAIN-BODY>

<MAIN-BODY> → CURLYOPEN <DECL-STMT> <ASSIGN-STMT> CURLYCLOSE

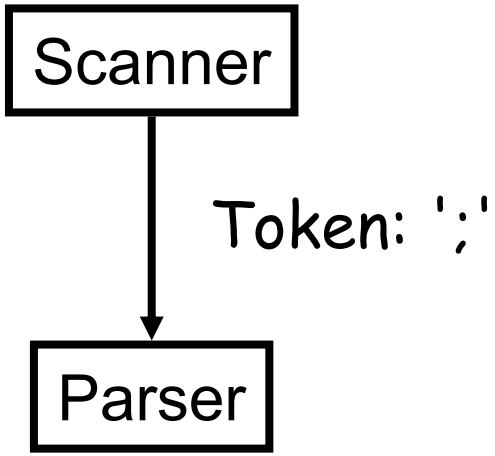
<DECL-STMT> → <TYPE>VAR<VAR-LIST>;

<VARLIST> → , VAR <VARLIST>

<VARLIST> → NULL

# Scanner/Parser

```
main() {
   int a,b;
   a = b;
}
```



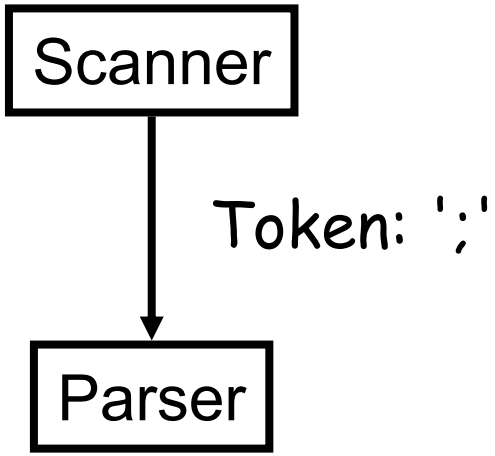

<C-PROG> → MAIN OPENPAR <PARAMETERS> CLOSEPAR <MAIN-BODY>

<MAIN-BODY> → CURLYOPEN <DECL-STMT> <ASSIGN-STMT> CURLYCLOSE

<DECL-STMT> → <TYPE>VAR<VAR-LIST>;

# Scanner/Parser

```
main() {
   int a,b;
   a = b;
}
```



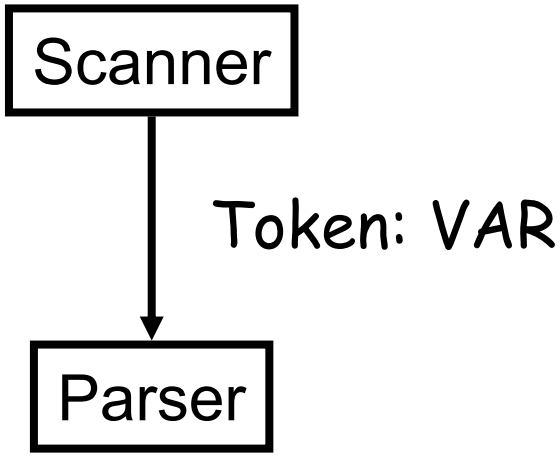

<C-PROG> → MAIN OPENPAR <PARAMETERS> CLOSEPAR <MAIN-BODY>

<MAIN-BODY> → CURLYOPEN <DECL-STMT> <ASSIGN-STMT> CURLYCLOSE

<DECL-STMT> → <TYPE>VAR<VAR-LIST>;

# Scanner/Parser

```
main() {
  int a,b;
  a = b;
}
```




<C-PROG> → MAIN OPENPAR <PARAMETERS> CLOSEPAR <MAIN-BODY>

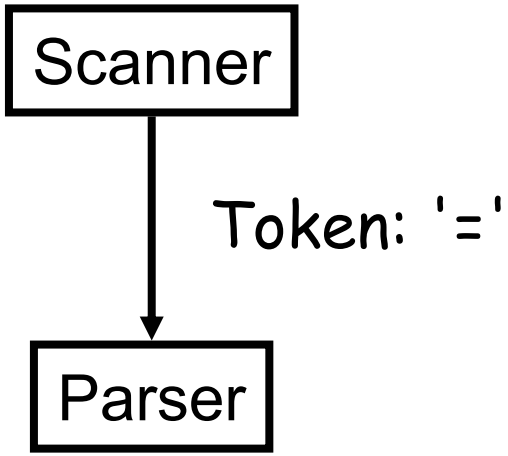
<MAIN-BODY> → CURLYOPEN <DECL-STMT> <ASSIGN-STMT> CURLYCLOSE

<ASSIGN-STMT> → VAR = <EXPR>;

<EXPR> → VAR

# Scanner/Parser

```
main() {
  int a,b;
  a = b;
}
```




<C-PROG> → MAIN OPENPAR <PARAMETERS> CLOSEPAR <MAIN-BODY>

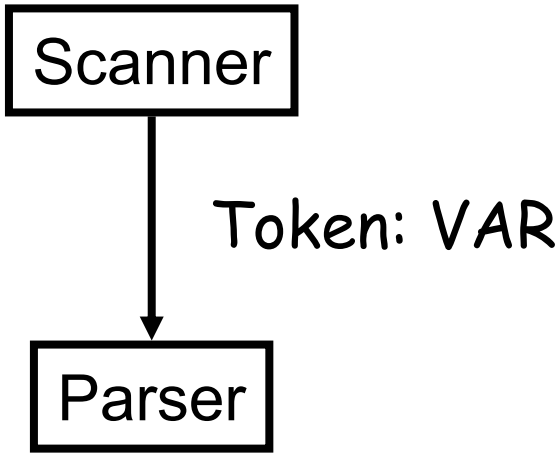
<MAIN-BODY> → CURLYOPEN <DECL-STMT> <ASSIGN-STMT> CURLYCLOSE

<ASSIGN-STMT> → VAR = <EXPR>;

<EXPR> → VAR

# Scanner/Parser

```
main() {
  int a,b;
  a = b;
}
```




<C-PROG> → MAIN OPENPAR <PARAMETERS> CLOSEPAR <MAIN-BODY>

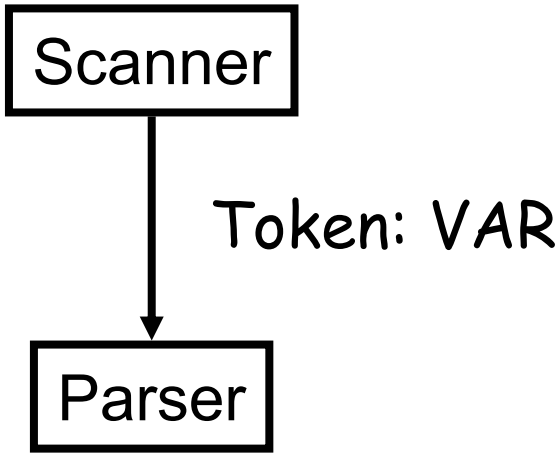
<MAIN-BODY> → CURLYOPEN <DECL-STMT> <ASSIGN-STMT> CURLYCLOSE

<ASSIGN-STMT> → VAR = <EXPR>;

<EXPR> → VAR

# Scanner/Parser

```
main() {
  int a,b;
  a = b;
}
```




<C-PROG> → MAIN OPENPAR <PARAMETERS> CLOSEPAR <MAIN-BODY>

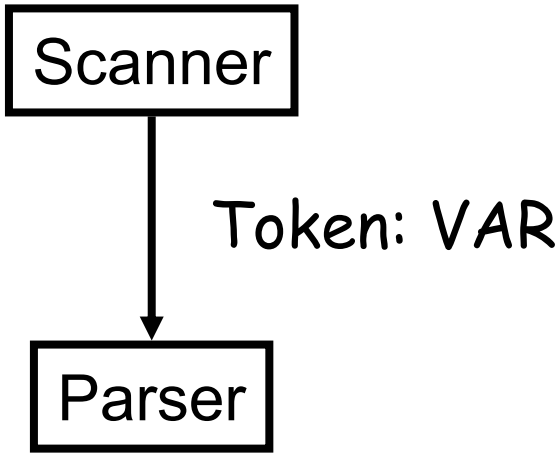
<MAIN-BODY> → CURLYOPEN <DECL-STMT> <ASSIGN-STMT> CURLYCLOSE

<ASSIGN-STMT> → VAR = <EXPR>;

<EXPR> → VAR

# Scanner/Parser

```
main() {
  int a,b;
  a = b;
}
```




<C-PROG> → MAIN OPENPAR <PARAMETERS> CLOSEPAR <MAIN-BODY>

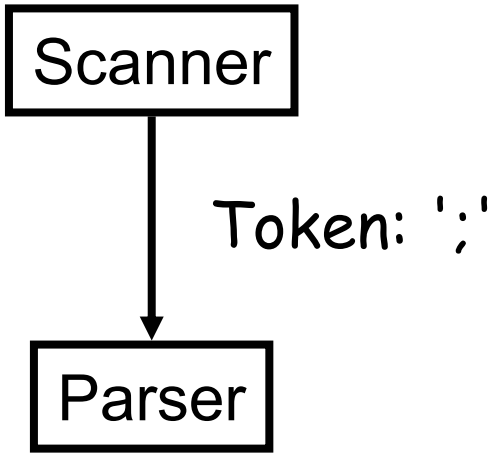
<MAIN-BODY> → CURLYOPEN <DECL-STMT> <ASSIGN-STMT> CURLYCLOSE

<ASSIGN-STMT> → VAR = <EXPR>;

<EXPR> → VAR

# Scanner/Parser

```
main() {
  int a,b;
  a = b;
}
```



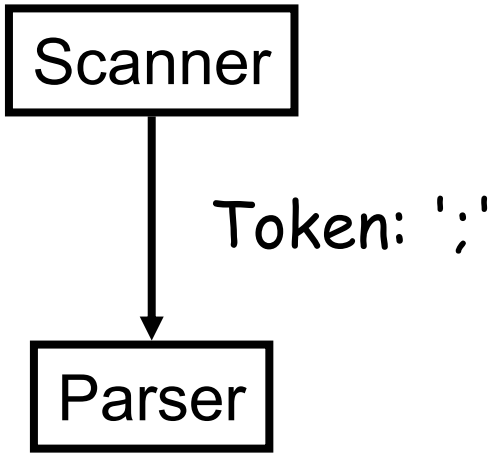

<C-PROG> → MAIN OPENPAR <PARAMETERS> CLOSEPAR <MAIN-BODY>

<MAIN-BODY> → CURLYOPEN <DECL-STMT> <ASSIGN-STMT> CURLYCLOSE

<ASSIGN-STMT> → VAR = <EXPR>;

# Scanner/Parser

```
main() {
  int a,b;
  a = b;
}
```



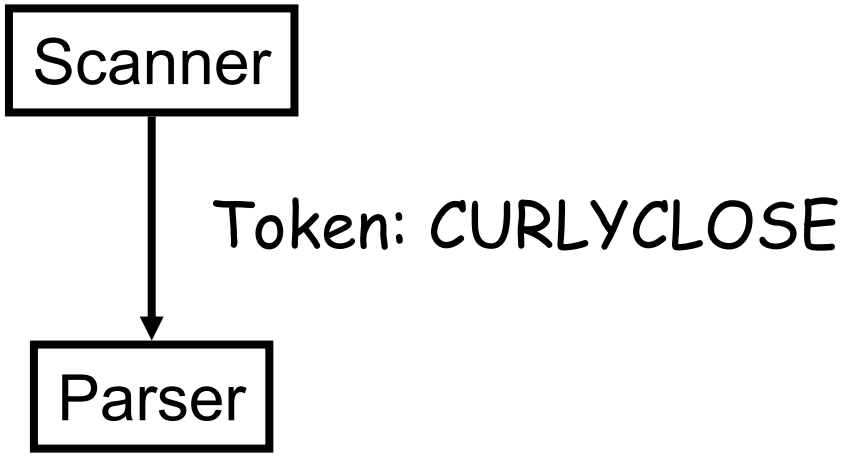

<C-PROG> → MAIN OPENPAR <PARAMETERS> CLOSEPAR <MAIN-BODY>

<MAIN-BODY> → CURLYOPEN <DECL-STMT> <ASSIGN-STMT> CURLYCLOSE

<ASSIGN-STMT> → VAR = <EXPR>;

# Scanner/Parser

```
main() {
  int a,b;
  a = b;
}
```




<C-PROG> → MAIN OPENPAR <PARAMETERS> CLOSEPAR <MAIN-BODY>

<MAIN-BODY> → CURLYOPEN <DECL-STMT> <ASSIGN-STMT> CURLYCLOSE

# Primjer (leksička i sintaktička analiza)

- Program GCD (Pascal):

```
program gcd(input, output);
var i, j : integer;
begin
    read(i, j);
    while i <> j do
        if i > j then i := i - j
        else j := j - i;
    writeln(i)
end.
```

# Leksička analiza

- Tokeni u programu GCD:

```
program   gcd     (       input   ,       output  )       ;
var       i       ,       j       :       integer ;      begin
read      (       i       ,       j       )       ;      while
i         <>      j       do      if      i       >      j
then      i       :=      i       -       j       else   j
:=        j       -       i       ;       writeln (      i
)         end     .
```

# KS gramatika i parsiranje

- KS gramatika za Pascal programe:

*program* ⟶ `PROGRAM id ( id` *more_ids* `) ;` *block* `.`

where

*block* ⟶ *labels constants types variables subroutines* `BEGIN` *stmt more_stmts* `END`

and

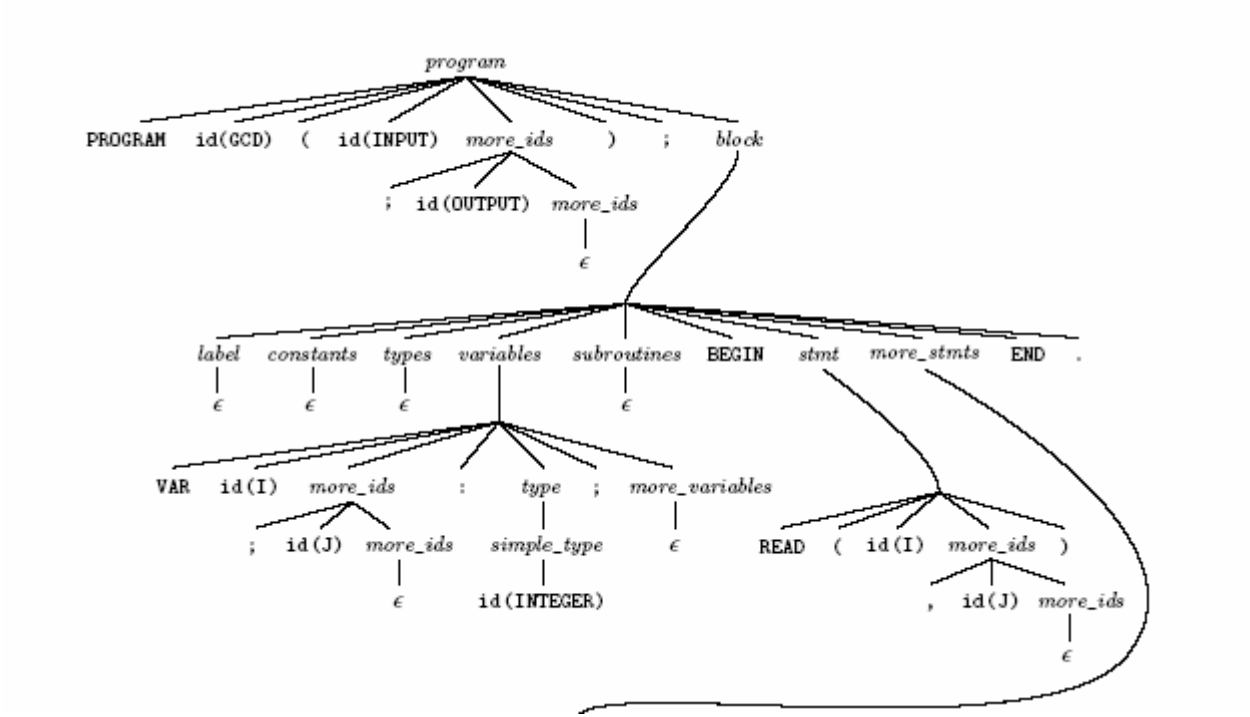
*more_ids* ⟶ `, id` *more_ids*

or

*more_ids* ⟶ $\epsilon$

# KS gramatika i parsiranje

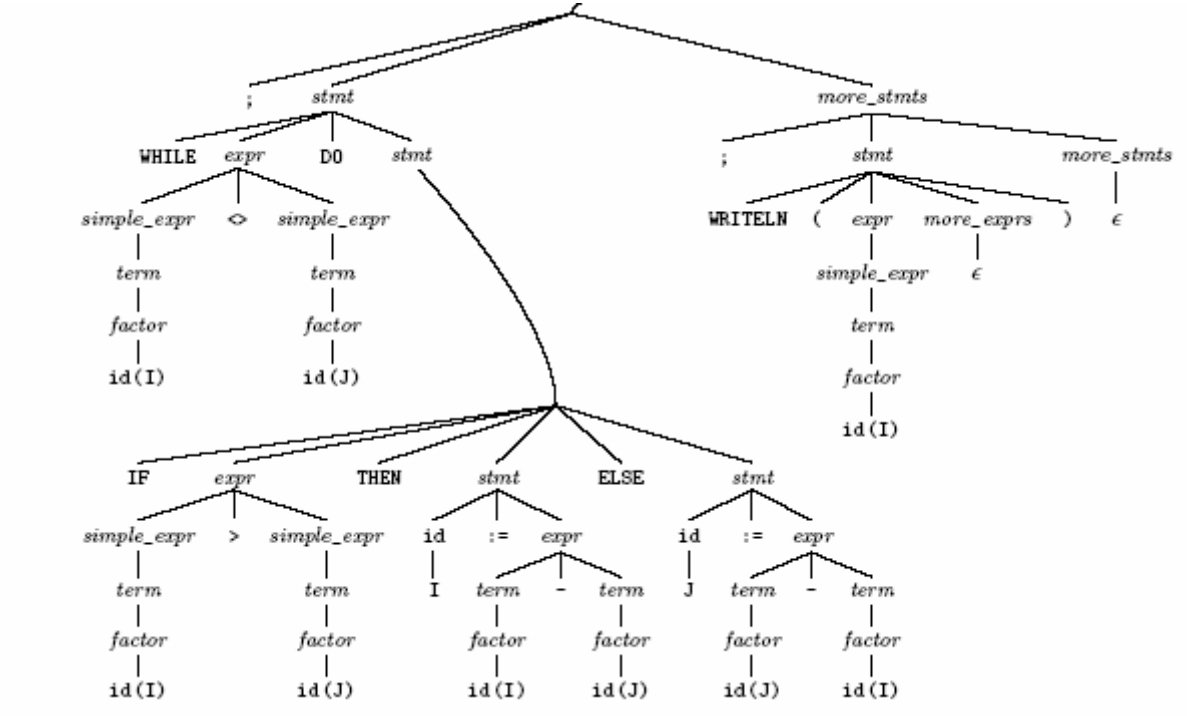
- Stablo izvoda za program GCD:




Na idućem slideu

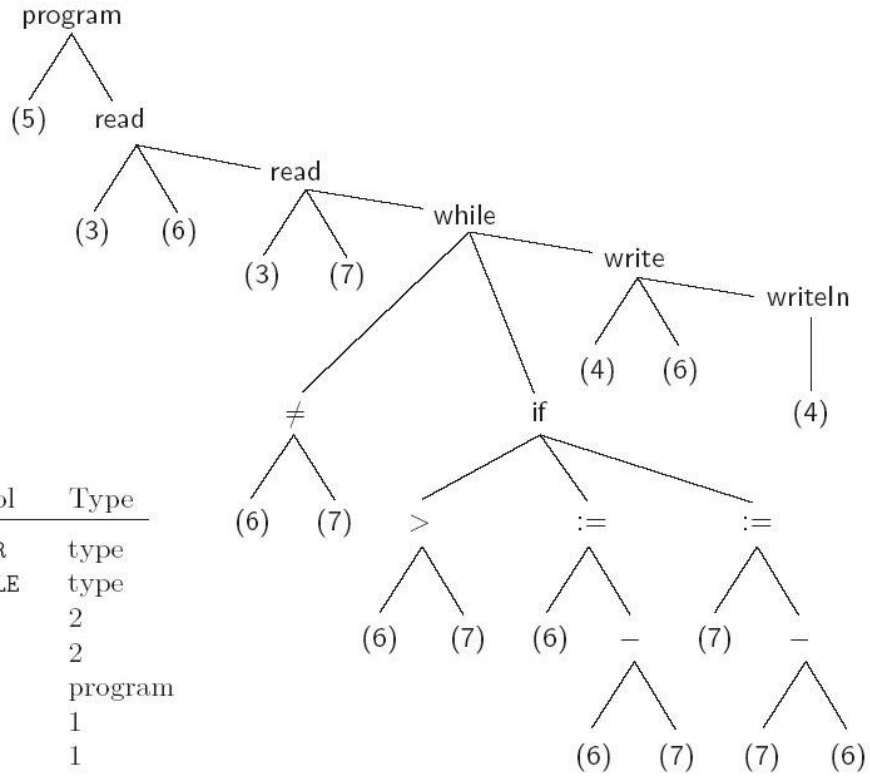
# KS gramatika i parsiranje

- Stablo izvoda za program GCD:



# KS gramatika i parsiranje

- Apstraktno sintaksno stablo za program GCD:




| Index | Symbol | Type |
|---|---|---|
| 1 | INTEGER | type |
| 2 | TEXTFILE | type |
| 3 | INPUT | 2 |
| 4 | OUTPUT | 2 |
| 5 | GCD | program |
| 6 | I | 1 |
| 7 | J | 1 |

Figure 1.4: **Syntax tree and symbol table for the GCD program.**

# Leksička analiza

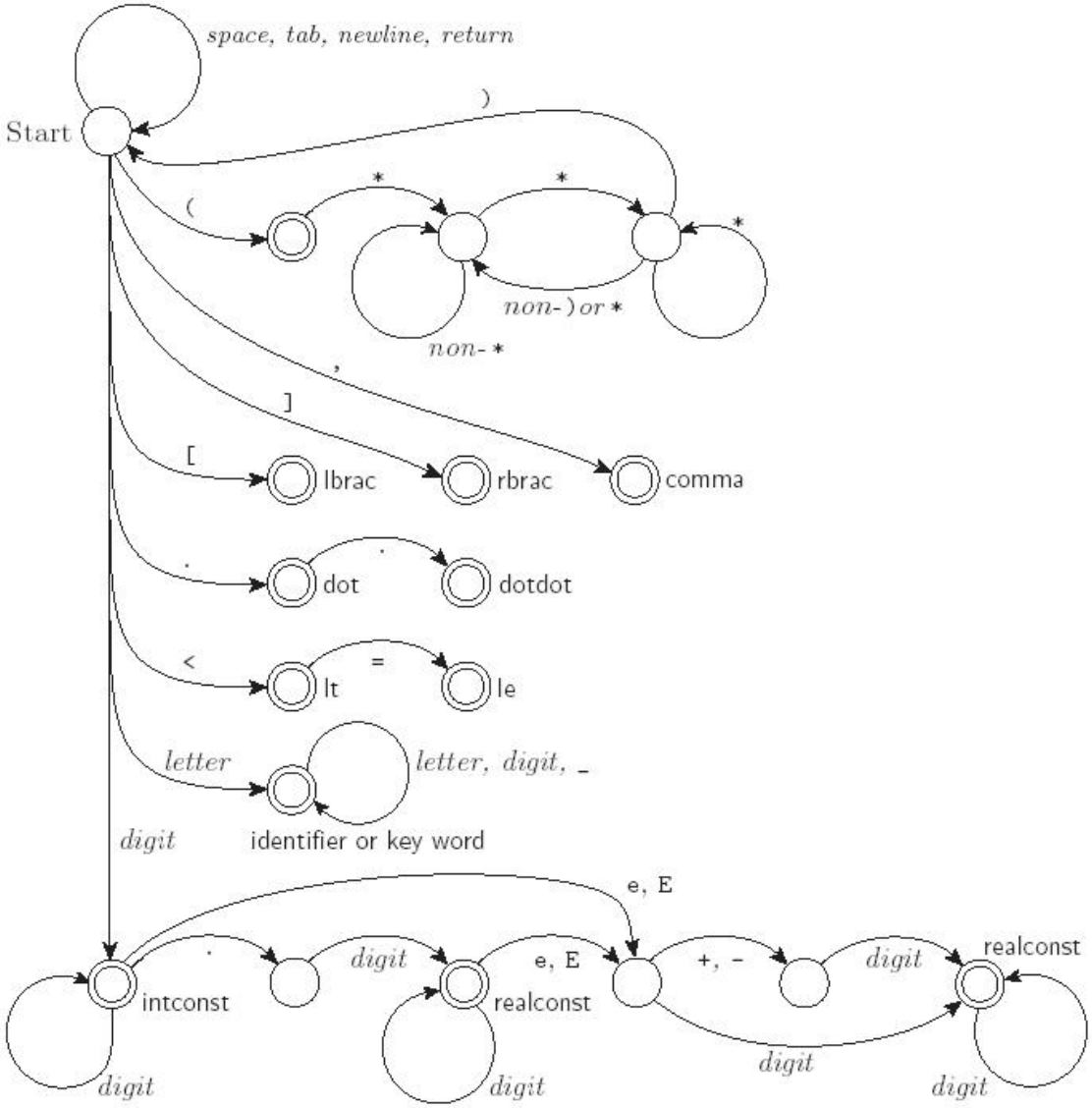
# Regularni izrazi

- Leksička struktura tokena opisuje se regularnim izrazima:
- Primjer (cijeli broj bez predznaka u Pascalu):

$$
\begin{aligned}
\mathit{digit} &\longrightarrow \texttt{0} \mid \texttt{1} \mid \texttt{2} \mid \texttt{3} \mid \texttt{4} \mid \texttt{5} \mid \texttt{6} \mid \texttt{7} \mid \texttt{8} \mid \texttt{9} \\
\mathit{unsigned\_integer} &\longrightarrow \mathit{digit}\ \mathit{digit}* \\
\mathit{unsigned\_number} &\longrightarrow \mathit{unsigned\_integer}\ (\ (\ \texttt{.}\ \mathit{unsigned\_integer}\ )\mid \epsilon\ ) \\
&\qquad\qquad (\ (\ (\ \texttt{e} \mid \texttt{E}\ )\ (\ \texttt{+} \mid \texttt{-} \mid \epsilon\ )\ \mathit{unsigned\_integer}\ ) \mid \epsilon\ )
\end{aligned}
$$

# Scanner

- Iz danog regularnog izraza moguće je automatski generirati konačni automat



# Problem

- Što u ovakvom slučaju?

```
void f (float* p, float y)
{
    float x;
    x = y/*p;
}
```

(/* je delimiter za komentare u C-u)

# Lookahead

- Kod nekih jezika odluku o tipu tokena nije moguće donijeti samo na temelju početnog prefiksa tokena
- Još jedan primjer: FORTRAN
  - Varijable ne trebaju biti eksplicitno deklarirane
  - Implicitno tipiziranje u ovisnosti u prvom karakteru
  - Realni brojevi počinju s A-H ili O-Z
  - Cijeli brojevi počinju s I-N
  - (Dizajneri jezika željeli oponašati standardne matematičke konvencije)

# Lookahead

- Tipičan izgled petlje u FORTRAN-u:

```
      DO 10 I = 1,10000
        ...
        ...
   10 CONTINUE
```

- Ekvivalent u C-u:

```
      for (i=1; i<=10000; ++i) {
        ...
      }
```

# Lookahead

- Sve praznine se ignoriraju, pa je

```
DO 10 I = 1, 100000
```

- ekvivalentno

```
DO10I=1,100000
```

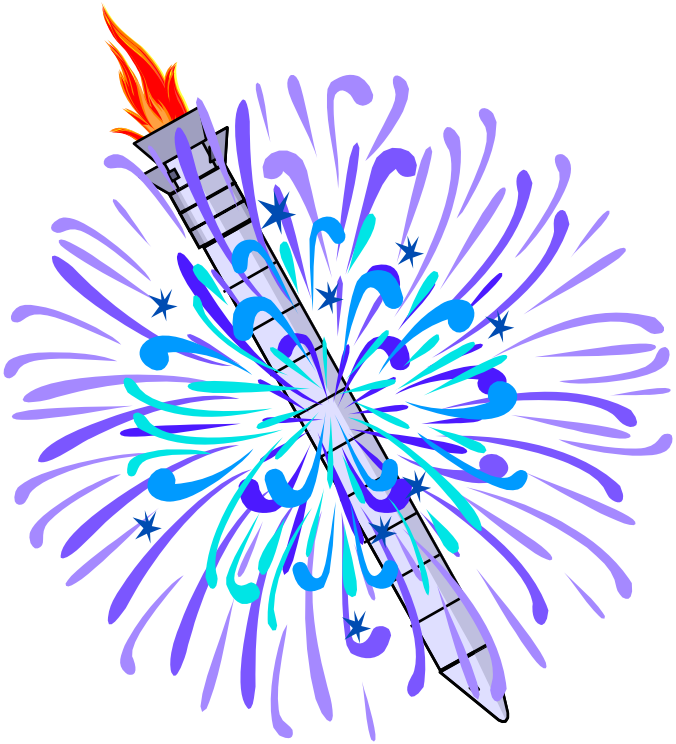
- što je pak različito od

```
DO 10 I = 1. 100000
```

tj.

```
DO10I=1.100000
```

# Rezultat



# Sintaktička analiza

# Kontekstno slobodni jezici

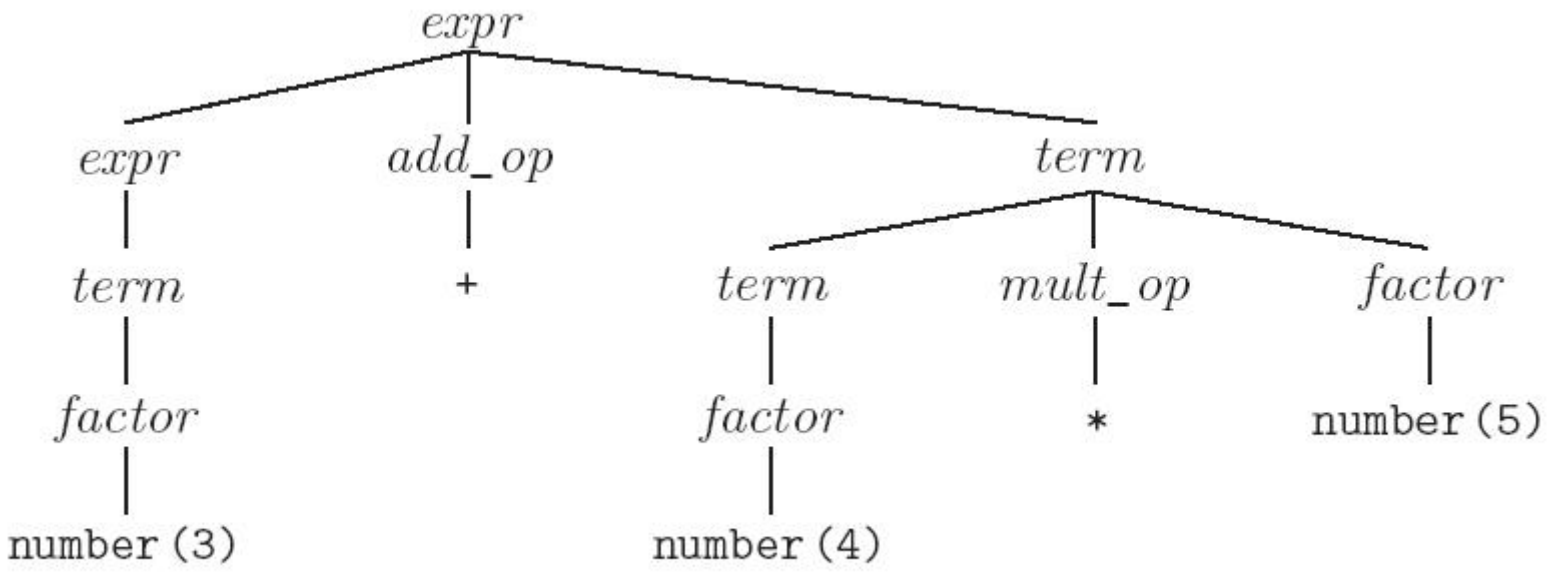
- Primjer: Gramatika aritmetičkih izraza

1. *expr* ⟶ *term* | *expr add_op term*
2. *term* ⟶ *factor* | *term mult_op factor*
3. *factor* ⟶ `id` | `number` | `-` *factor* | `(` *expr* `)`
4. *add_op* ⟶ `+` | `-`
5. *mult_op* ⟶ `*` | `/`

# Kontekstno slobodni jezici

- Primjer: Stablo izvoda za izraz

  **3 + 4 * 5**



# Još jedan primjer – XML i DTD

- DTD je KS gramatika za opis strukture neke klase XML dokumenata
- Primjer:

```
<?xml version="1.0"?>
<!DOCTYPE oldjoke [
  <!ELEMENT oldjoke (burns+, allen, applause?)>
  <!ELEMENT burns (#PCDATA | quote)*>
  <!ELEMENT allen (#PCDATA | quote)*>
  <!ELEMENT quote (#PCDATA)*>
  <!ELEMENT applause EMPTY>
]>
<oldjoke>
  <burns>Say <quote>goodnight</quote>,
  Gracie.</burns>
  <allen><quote>Goodnight, Gracie.</quote></allen>
  <applause />
</oldjoke>
```

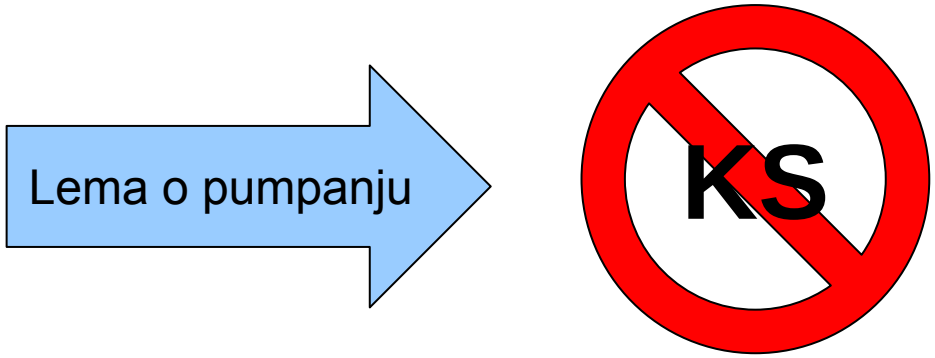
# Kontekstno slobodni jezici

- L= {w \in ASCII* | w je sintaktički korektan C program}
- Konstruiramo C program u ovisnosti o n iz leme o pumpanju:

n=4

```
void main(void)
    {
    int v1;
    int v2;
    int v3;
    int v4;

    v1 = 0;
    v2 = 0;
    v3 = 0;
    v4 = 0;
    }
```



# Kontekstno slobodni jezici

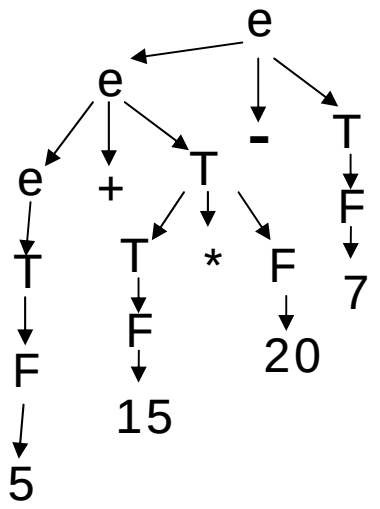
- Klasa KS jezika nije dovoljna za opis svih pravila programskih kezika:
  - "svi identifikatori moraju biti deklarirani prije upotrebe":
    L = { w # w | w \in (a+b)* }
  - "broj formalnih parametara u deklaraciji funkcije podudara se s brojem argumenata kod funkcijskog poziva":
    L = { a^m b^n c^m d^n | m, n >= 1 }

# Kontekstno slobodni jezici

- "Klasa KS jezika od velike je važnosti u računarstvu"
- Dva (kontradiktorna) problema:
  1. KS jezici nisu dovoljno snažni za opis svih pravila programskih jezika (npr. tipovnog sustava)
  2. Opis gotovo svih sintaktičkih pravila programskih jezika u pravilu ne zahtjeva punu ekspresivnost KS jezika
- Želimo parser koji parsira ulazni program duljine n u vremenu O(n)
- U praksi:
  - restriktivnije gramatike – klase LL(k) i LR(k)
  - ekspresivnije gramatike – atribuirane, dvorazinske, W-, DCG (Prolog)

# Top-down parsiranje

- Počevši od startnog neterminalnog simbola trebamo naći najljeviji izvod




**Gramatika:**

E : : = E + T | E – T | T
T : : = T * F | T / F | F
F : : = n

**Najljeviji izvod za 5+15*20-7:**

**E➔E-T➔E+T-T➔T+T-T**
**➔F+T-T➔5+T-T➔5+T*F-T➔5+F*F-T➔5+15*F-T**
**➔5+15*20-T➔5+15*20-F➔5+15*20-7**

# Prediktivno parsiranje

- LL(k) parser - s lijeva na desno, najljeviji izvod, k-simbola lookahead
  - Odluka o odabiru produkcije gramatike donosi se na temelju provjere narednih k tokena
  - LL(k) gramatika – gramatika koju je moguće parsirati LL(k) parserom
- LL(1) parser – produkcija gramatike se bira samo na temelju idućeg tokena

**Gramatika:**

E : : = E + T | E – T | T
T : : = T * F | T / F | F
F : : = n

**Svaka produkcija za E započinje s E -> nije LL(K)**

**Ekvivalentna LL(1) gramatika :**

**Gramatika:**

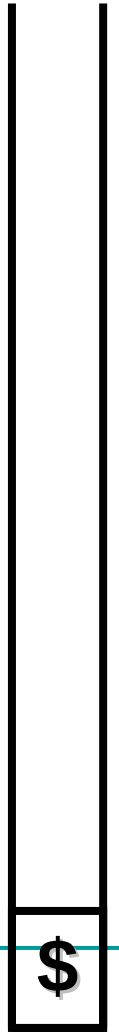
E : : = TE’
E’ : : = + TE’ | - TE’ | ε
T : : = FT’
T’ : : = *FT’ | / FT’ | ε
F : : = n

# LL(1) parsiranje - Primjer

**Gramatika**
**S → ( S ) S | ε**

**Ulaz**
**()$**

- $ - oznaka dna stoga i kraja ulazne riječi



# LL(1) parsiranje - Primjer

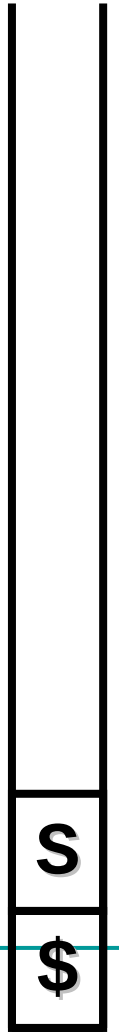
**Gramatika**

**S → ( S ) S | ε**

**Ulaz**

**()$**

- Stavimo početni simbol S na stog



# LL(1) parsiranje - Primjer

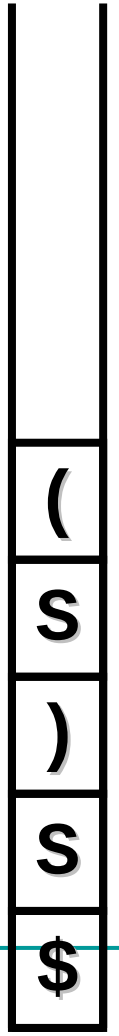
**Gramatika**

**S → ( S ) S | ε**

**Ulaz**

**()$**

- Expand: **S → ( S ) S**



# LL(1) parsiranje - Primjer

**Gramatika**

**S → ( S ) S | ε**

**Ulaz**

**()$**

- Match

# LL(1) parsiranje - Primjer

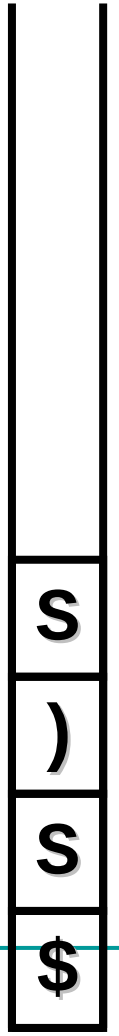
**Gramatika**

**S → ( S ) S | ε**

**Ulaz**

**)$**

- Expand: **S → ε**



# LL(1) parsiranje - Primjer

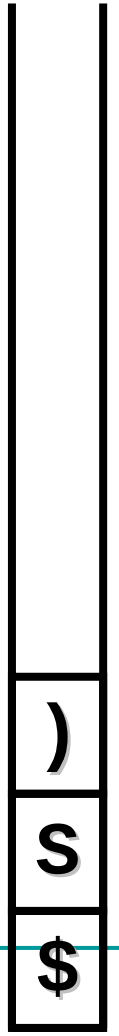
**Gramatika**

**S → ( S ) S | ε**

**Ulaz**

**)$**

- Expand: **S → ε**



# LL(1) parsiranje - Primjer

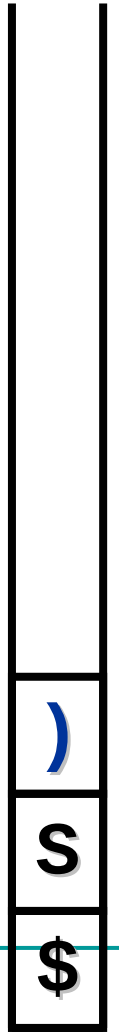
**Gramatika**

**S → ( S ) S | ε**

**Ulaz**

**)$**

- Match



# LL(1) parsiranje - Primjer

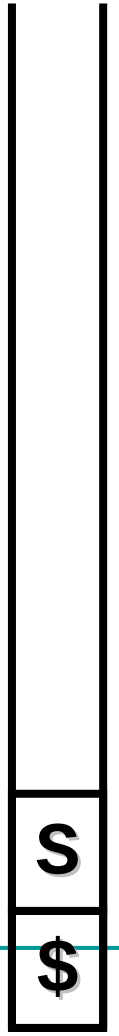
**Gramatika**

**S → ( S ) S | ε**

**Ulaz**

**$**

- Expand: **S → ε**

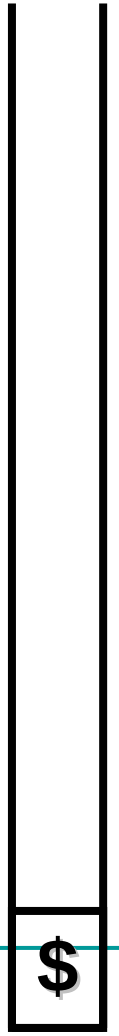


# LL(1) parsiranje - Primjer

**Gramatika**
**S → ( S ) S | ε**

**Ulaz**
**$**

- Expand: **S → ε**



# LL(1) parsiranje - Primjer

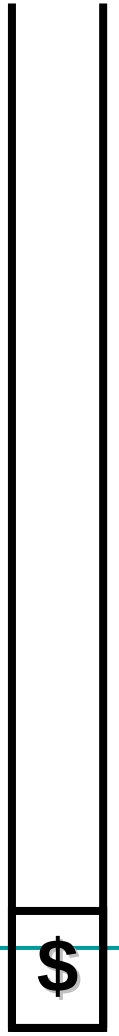
**Gramatika**

**S → ( S ) S | ε**

**Ulaz**

**$**

- Parsiranje uspješno završilo!

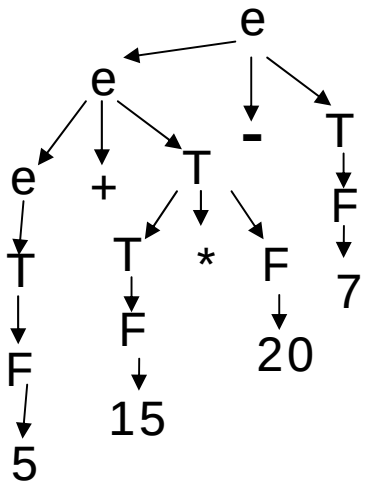


# Bottom-up parsiranje

- Počevši od ulaznog stringa, trebamo reducirati ulazni string na početni neterminalni simbol
(ekvivalentno obrnutom najdesnijem izvodu)

**Gramatika:**

E : : = E + T | E – T | T
T : : = T * F | T / F | F
F : : = n




**Najdesniji izvod za 5+15*20-7:**

**E➔E-T➔E-F➔E-7➔E+T-7➔E+T*F-7**
**➔E+T*20-7➔E+F*20-7➔E+15*20-7**
**➔T+15*20-7➔F+15*20-7➔5+15*20-7**

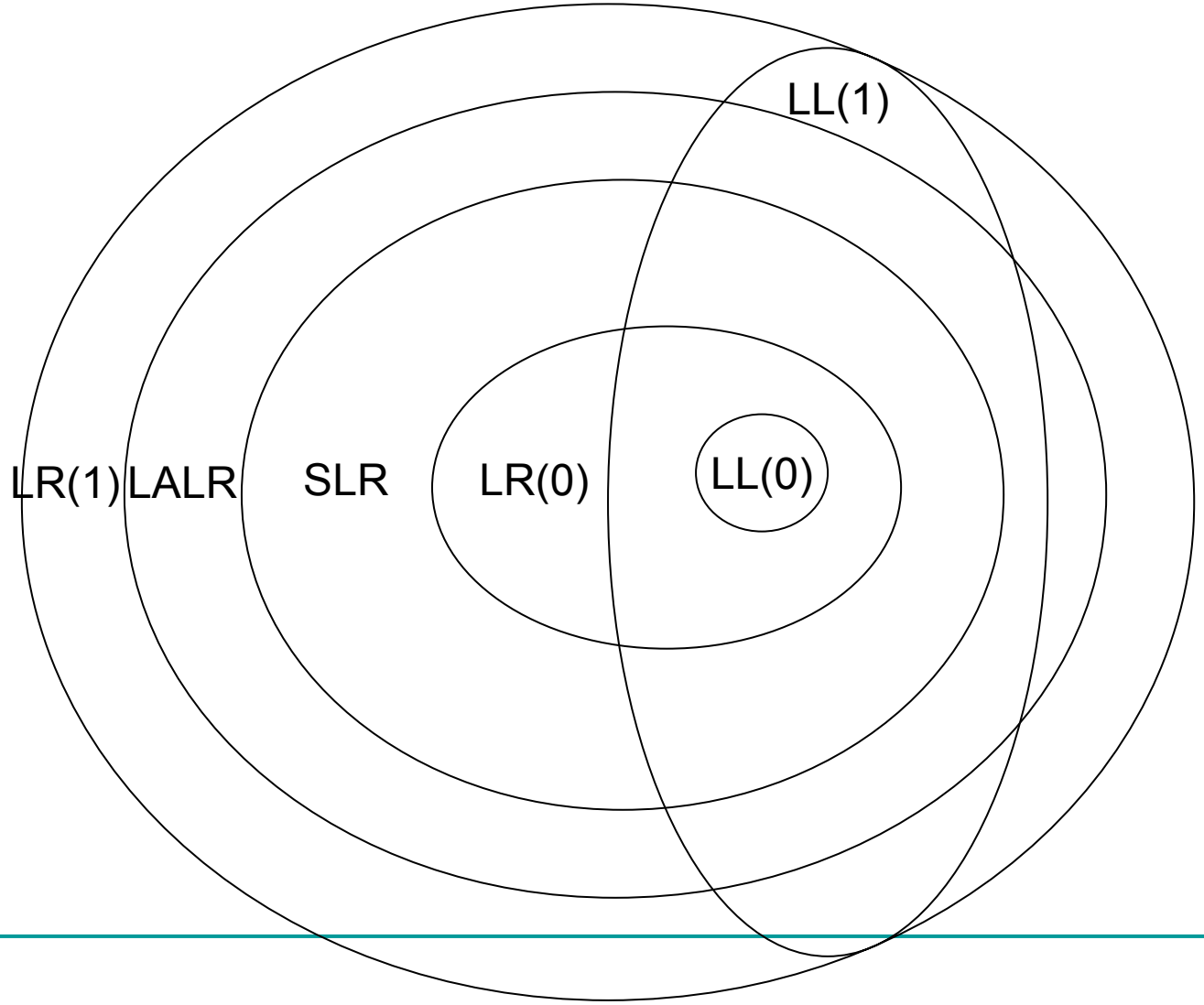
**Bottom-up parsiranje:**

**5+15*20-7➔F+15*20-7➔T+15*20-7**
**➔E+15*20-7➔E+F*20-7➔E+T*20-7**
**➔E+T*F-7➔E+T-7➔E-7➔E-F➔E-T➔E**

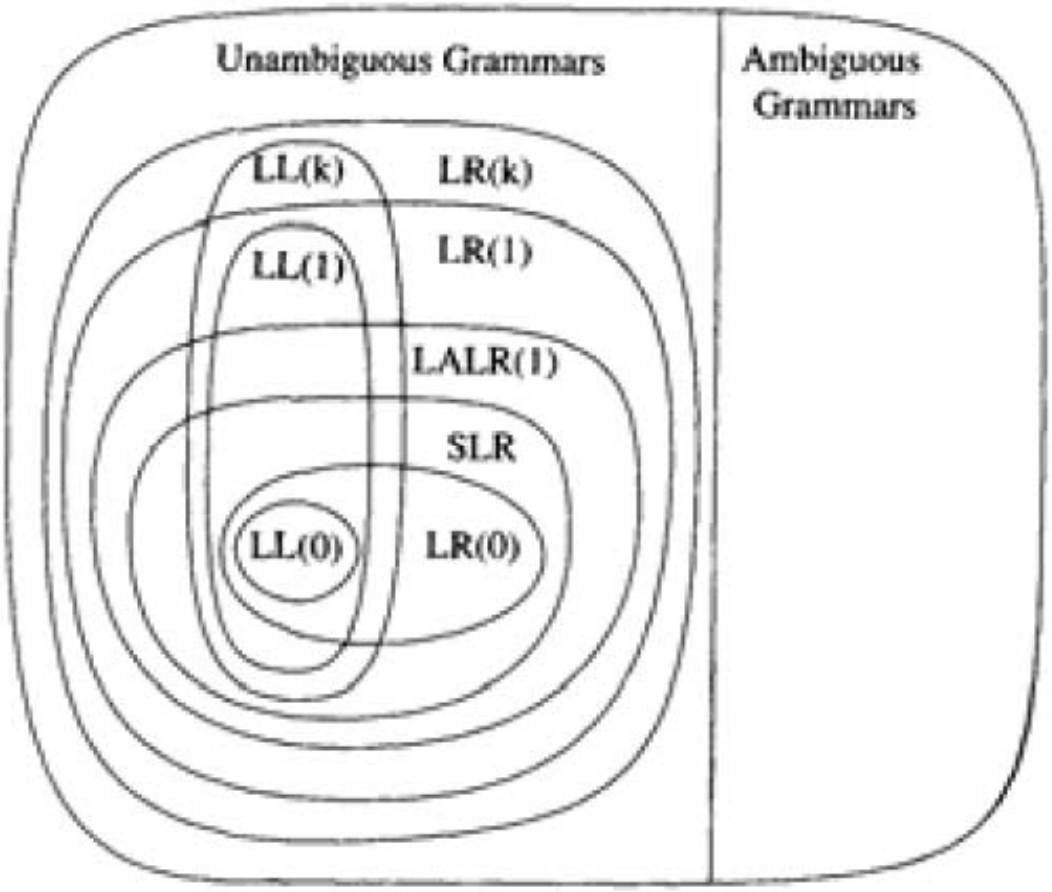
# LR(k) parsiranje

- LR(k) parser - s lijeva na desno, najdesniji izvod, k simbola lookahead
  - Odluke donose na temelju provjere narednih k tokena
  - Koriste konačne automate za konfiguraciju akcija (shift ili reduce)
    - Stanja automata pamte simbole koji su već na stogu
    - Svaki par (stanje, token) jedinstveno određuje iduću akciju
- Zašto koristiti LR parsere?
  - Prepoznaju širu klasu KS jezika od prediktivnih LL(k) parsera
  - Efikasna implementacija, nema backtrackinga
- LR(k) vs LL(k):
  - LL parseri – pogodni za "ručnu" implementaciju (jednostavan programski kod)
  - LR parseri – bitno teža "ručna" implementacija; obično se koriste automatski generatori (npr. Yacc/Bison) -> kompliciran i "ružan" programski kod

# Odnos među klasama gramatika

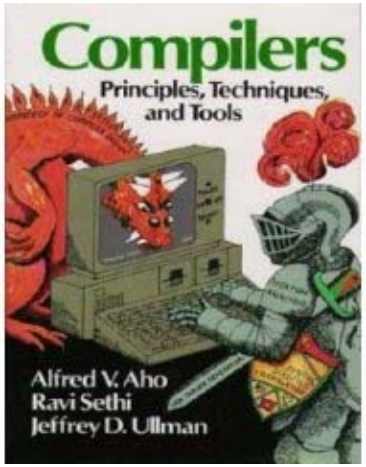


# Odnos među klasama gramatika

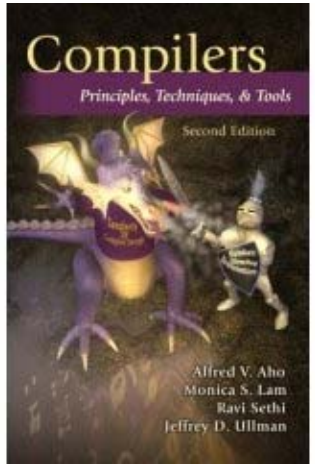


# The Dragon Book

- **Compilers: Principles, Techniques and Tools**
  - Alfred V. Aho, Ravi Sethi, Jeffrey D. Ullman (1st Edition)




  - Alfred V. Aho, Monica S. Lam, Ravi Sethi, Jeffrey D. Ullman (2nd Edition)

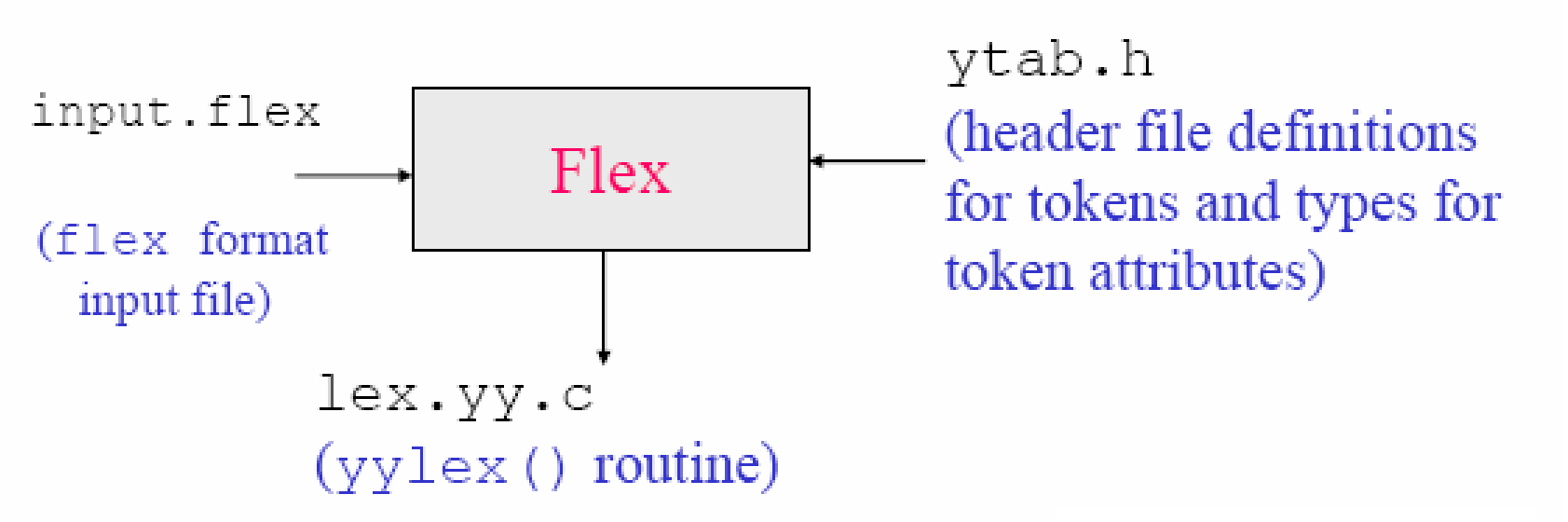


# Alati za generiranje lexera / parsera

# (F)lex

- (F)lex = (Fast) Lexical Analyzer Generator
- Neke implementacije:
  - Flex for Windows
    - http://gnuwin32.sourceforge.net/packages/flex.htm
  - JFlex
    - http://www.jflex.de/

# (F)lex

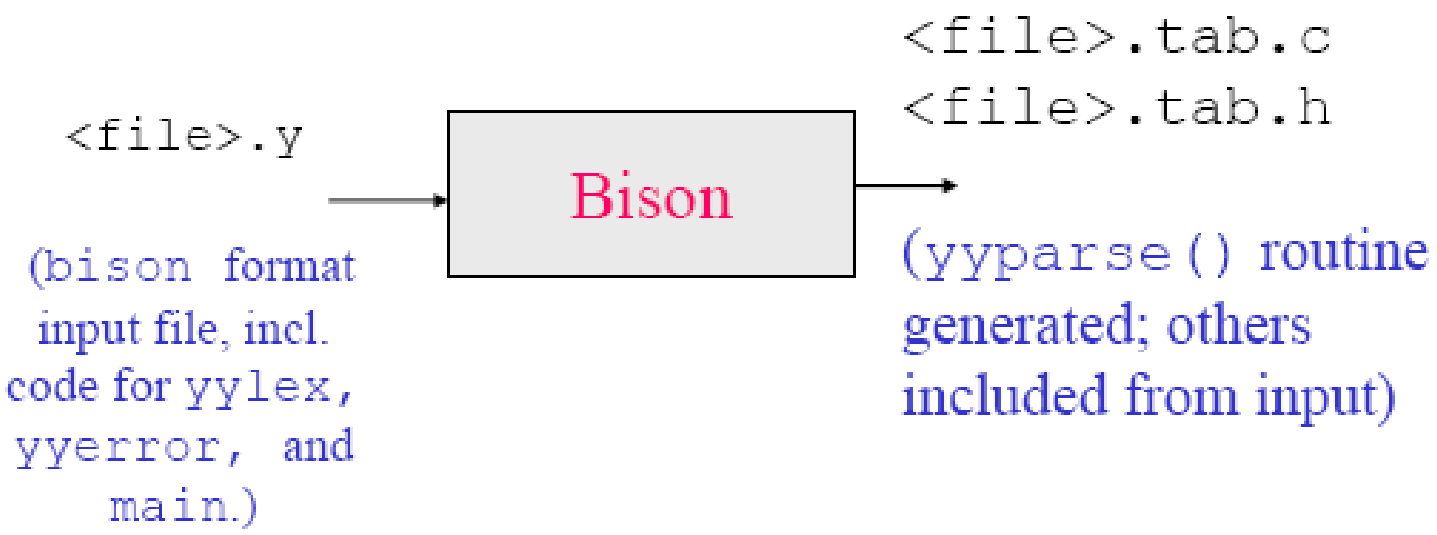



```
definitions
%%
rules
%%
user code
```

# Yacc/Bison

- Yacc = Yet Another Compiler-Compiler
  - generator LALR(1) parsera
- Bison – generator LALR(1) / GLR parsera (kompatibilan s Yacc-ovom sintaksom ulaza)
- Neke implementacije:
  - Bison for Windows
    - http://gnuwin32.sourceforge.net/packages/bison.htm
  - BYACC/J
    - http://byaccj.sourceforge.net/

# Yacc/Bison

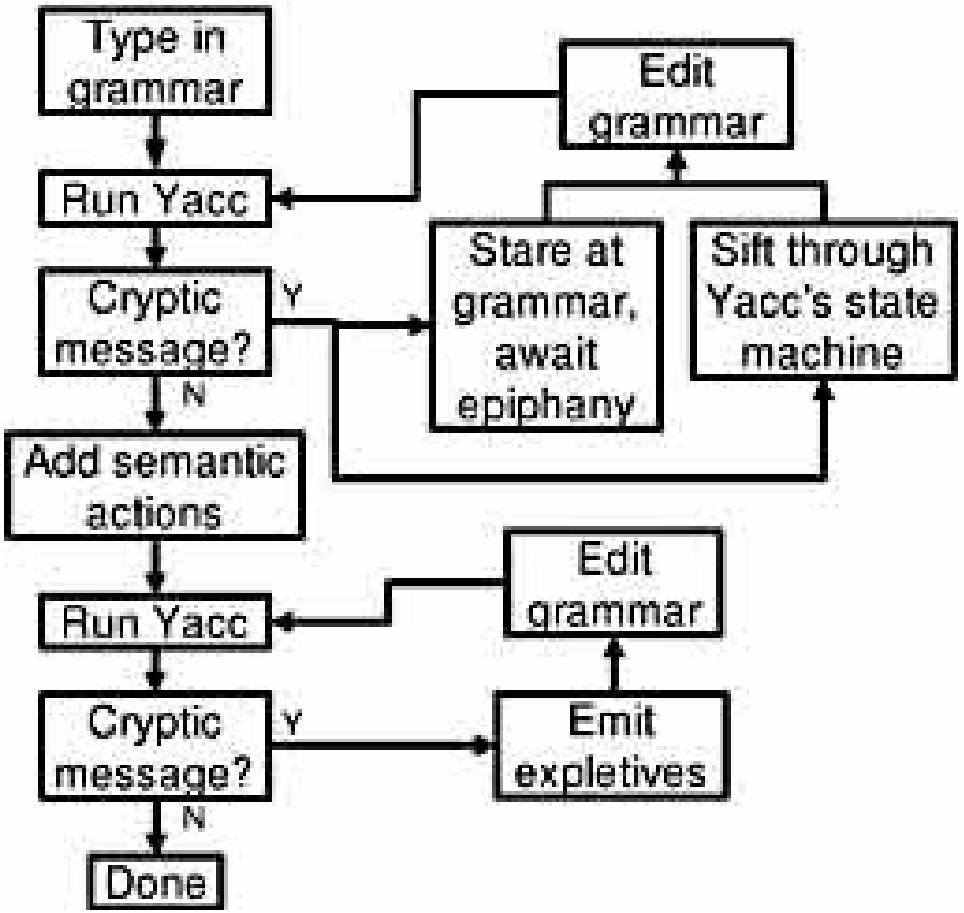

<file>.y
Bison
<file>.tab.c
<file>.tab.h
(bison format input file, incl. code for yylex, yyerror, and main.)
(yyparse() routine generated; others included from input)

# Yacc/Bison

The input file consists of three sections, separated by a line with just `%%' on it:

```
%{
C declarations (types, variables, functions,
                preprocessor commands)
%}
Bison declarations (grammar symbols,
                    operator precedence decl.,
                    attribute data type)
%%
Grammar rules
%%
Additional C code (incl. scanner yylex)
```

# Ciklus razvoja parsera s Yacc/Bison-om




http://www.acm.org/crossroads/xrds7-5/bison.html

# ANTLR

- ANTLR = ANother Tool for Language Recognition
  - generator lexera i LL(k) parsera
    - zapravo LL(*) s linearnom aproksimacijom lookaheada i sintaktičkim/semantičkim predikatima
    - zajednička sintaksa za specifikaciju lexera i parsera
- Linkovi:
  - ANTLR
    - http://www.antlr.org/
  - ANTLR plugin za Eclipse
    - http://antlreclipse.sourceforge.net/